Міністерство освіти й науки України

Національний технічний університет України

«Київський політехнічний інститут ім. Ігоря Сікорського»

Кафедра автоматизації проектування енергетичних процесів і систем

ЗВІТ

Про виконання лабораторної роботи №1

З дисципліни «Комп’ютерне моделювання та оптимізація»

**Моделювання стаціонарних систем**

TI-92 Черноусова Дениса

Перевірив проф. д.т.н. Шушура О. М.

Київ – 2021

g = 3

k = 2

g – остання цифра у номері студентського квитка, а k – передостання.

1. Для функції *f (x) = sin (x) + x* скласти таблицю значень, заповнивши таблицю 1. На основі складеної таблиці побудувати інтерполяційні поліноми Лагранжа і Ньютона (для нерівних проміжків). Порівняти значення інтерполяційних поліномів в точках х = -3,8; -1,5; 1,7; зі значеннями точної функції, оцінити точність наближення.

Таблиця 1.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***x*** | ***-4*** | ***-2*** | ***-1*** | ***1*** |
| ***f(x)*** | **-3,243** | **-2,909** | **-1,841** | **1,841** |

де g – остання цифра у номері студентського квитка, а k – передостання.

1. Для функції, заданої таблицею 1.1, за допомогою поліномів Чебишева визначити вузли, на основі яких можлива побудова інтерполяційного полінома з мінімальною максимальною похибкою. У якості відрізка взяти [-4, 1]. Заповнити в цих вузлах нову таблицю значень для *f (x) = sin (x) + x* і побудувати по ній інтерполяційний поліном Лагранжа. Порівняти значення поліномів Лагранжа, отриманих в задачах 1 і 2 в точках х = -3,8; -1,5; 1,7; оцінити точність.
2. Для функції *f (x) = sin (x) + x* скласти таблицю значень, заповнивши таблицю 2.1. На основі складеної таблиці побудувати перший інтерполяційний поліном Ньютона (для рівновіддалених вузлів). Порівняти значення полінома в точках х = 2,7; -1,5; 0,8; зі значеннями точної функції, оцінити точність наближення.

Таблиця 2.1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ***X*** | ***-3*** | ***-2*** | ***-1*** | ***0*** |
| ***f(x)*** | **-3,141** | **-2,909** | **-1,841** | **0** |

1. Побудувати лінійний сплайн для функції, заданої таблицею 1.1. Обчислити наближені значення в точках х = -2,6;- 1,5; -0,1, оцінити точність.
2. За допомогою методу найменших квадратів побудувати наближення функції, заданої таблицею 2.1, лінійною функцією y = a + bx. Побудувати графік знайденої прямої, зобразити на ньому точки вихідної функції. Розрахувати наближене значення в точках х = -3,8; -1,5; 1,7; порівняти з точним значенням і значеннями поліному, отриманому в задачі 3.
3. За допомогою методу найменших квадратів побудувати наближення функції, заданої таблицею даних варіанту (наведені нижче), лінійною функцією **. Перед побудовою моделі перевірити дані на наявність помилок та виключити їх. Розрахувати наближене значення функції для значень аргументів з першого рядку таблиці. Номер варіанту дорівнює 6.4, де 3– остання цифра у номері студентського квитка +1.

*Варіант № 6.4*

Для мінімізації витрат на перевезення власник автопідприємства провів експеримент по встановленню залежності витрат палива на 100 км. від тиску в шинах автомобіля, ціни палива та ваги вантажу, що перевозиться. Результати експерименту наведені в таблиці 4. Вважаючи інші умови проведення перевезень однаковими, необхідно знайти залежність витрат на паливо від зазначених факторів.

Таблиця 4

Дані, отримані під час експерименту

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| № | Тиск в шинах, (атм.) | Вартість палива (грн.) | Вага вантажу, (т) | Витрати палива на перевезення на 100 км (грн.) |
| 1 | 4,04 | 24,05 | 3,76 | 351,10 |
| 2 | 4,36 | 24,06 | 2,41 | 351,36 |
| 3 | 3,43 | 25,50 | 1,96 | 347,74 |
| 5 | 4,97 | 21,77 | 1,54 | 349,18 |
| 6 | 3,13 | 23,33 | 3,01 | 348,44 |
| 7 | 3,50 | 21,73 | 3,58 | 352,93 |
| 8 | 4,69 | 25,28 | 1,34 | 346,11 |
| 9 | 4,27 | 23,93 | 2,60 | 349,22 |
| 10 | 3,03 | 25,11 | 3,09 | 338,15 |
| 11 | 3,98 | 24,46 | 1,76 | 338,80 |
| 12 | 3,52 | 22,99 | 2,23 | 351,71 |
| 13 | 5,00 | 21,84 | 3,32 | 345,63 |
| 14 | 4,89 | 24,57 | 3,17 | 328,31 |
| 15 | 5,15 | 23,58 | 1,25 | 346,76 |
| 16 | 2,16 | 24,04 | 1,39 | 346,09 |
| 18 | 2,38 | 23,13 | 4,52 | 347,54 |
| 19 | 3,33 | 24,20 | 2,32 | 353,28 |
| 20 | 5,09 | 24,47 | 3,71 | 341,21 |
| 21 | 4,50 | 24,76 | 2,08 | 349,96 |
| 22 | 4,19 | 25,51 | 1,46 | 342,54 |
| 24 | 4,30 | 23,78 | 2,68 | 340,60 |
| 27 | 4,84 | 23,80 | 2,93 | 343,35 |
| 28 | 4,47 | 23,19 | 2,34 | 351,46 |
| 29 | 3,54 | 23,05 | 2,30 | 353,53 |
| 30 | 3,64 | 23,61 | 2,50 | 336,24 |
| 31 | 3,42 | 23,01 | 4,20 | 349,76 |
| 32 | 5,54 | 23,96 | 2,31 | 332,60 |
| 33 | 4,58 | 25,41 | 2,15 | 342,55 |
| 34 | 2,93 | 25,23 | 2,53 | 339,24 |
| 37 | 3,84 | 23,04 | 3,32 | 316,80 |
| 38 | 4,33 | 24,60 | 1,44 | 339,64 |
| 39 | 3,58 | 24,04 | 1,87 | 341,23 |
| 41 | 4,68 | 22,95 | 2,28 | 349,13 |
| 42 | 4,26 | 23,24 | 4,22 | 336,21 |
| 43 | 4,69 | 23,27 | 3,70 | 346,00 |
| 45 | 3,03 | 23,97 | 3,09 | 351,39 |
| 46 | 3,03 | 23,00 | 2,22 | 349,32 |
| 47 | 4,26 | 23,73 | 1,76 | 345,76 |
| 48 | 2,72 | 23,94 | 2,14 | 339,84 |
| 49 | 3,27 | 25,03 | 2,32 | 349,67 |
| 50 | 2,83 | 23,87 | 2,70 | 346,62 |

**Алгоритми**

Інтерполяційний поліном Лагранжа



Інтерполяційний поліном Ньютона для нерівних проміжків

Поліном Чебишева

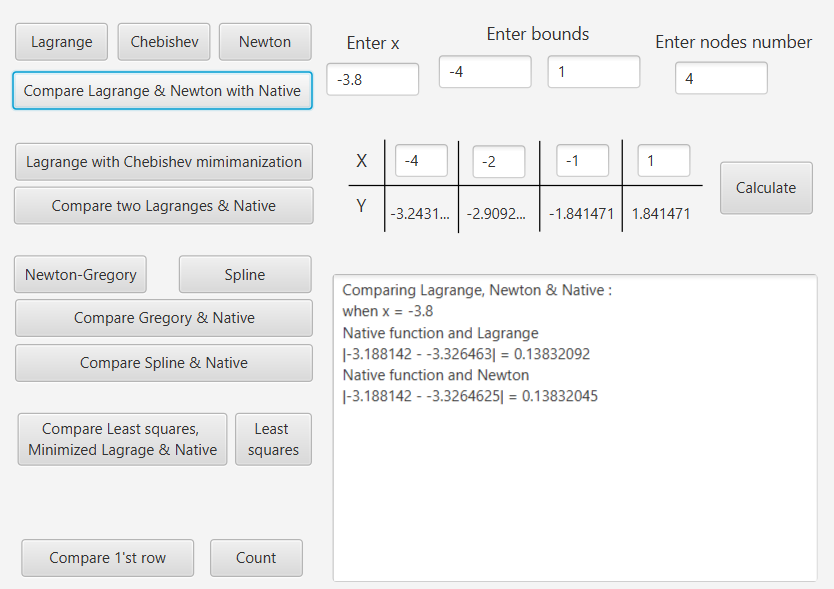
Інтерполяційний поліном Ньютона для рівновіддалених вузлів

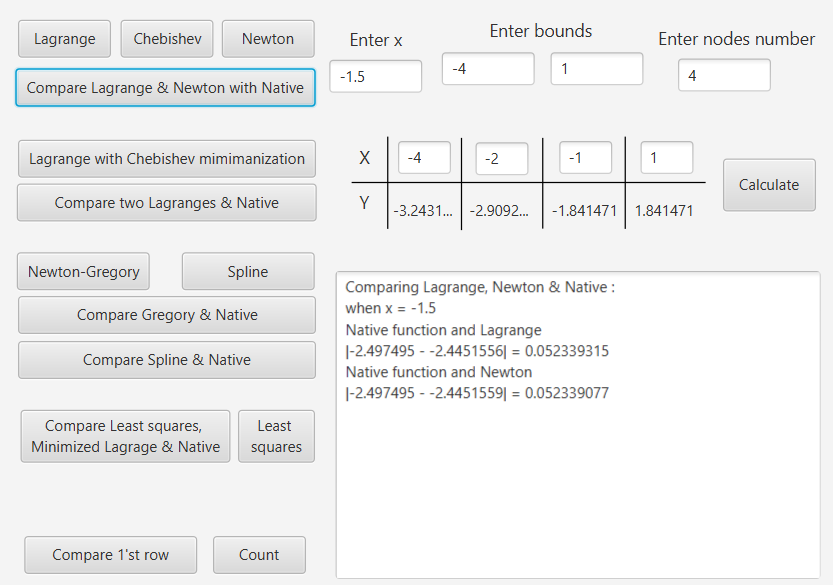
Лінійний сплайн

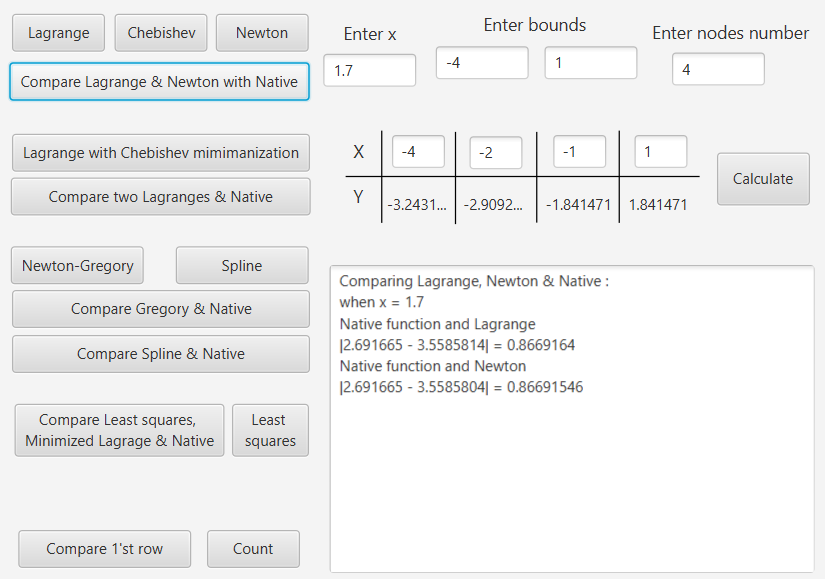
Метод Найменших Квадратів

**Результати роботи**

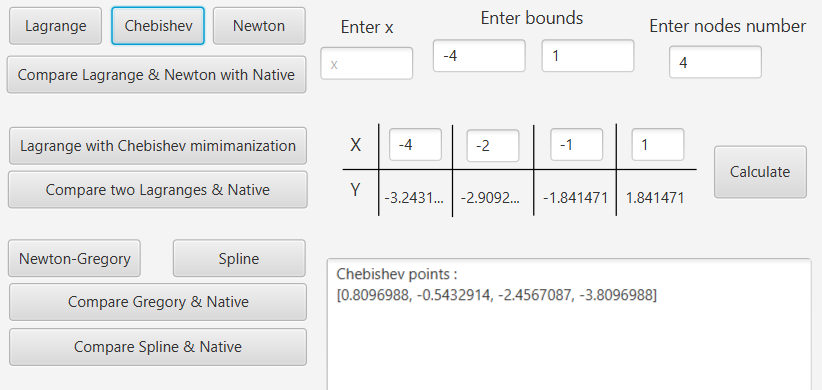
Завдання 1

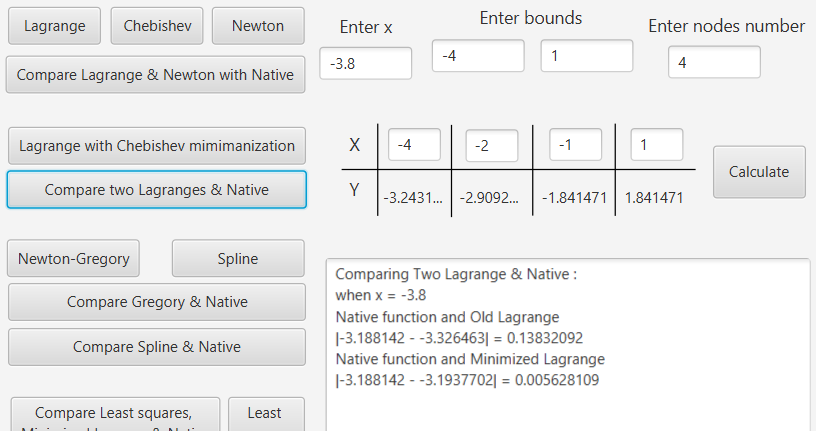
Порівнювання значень поліномів Лагранжа і Нюьтона з точним значенням

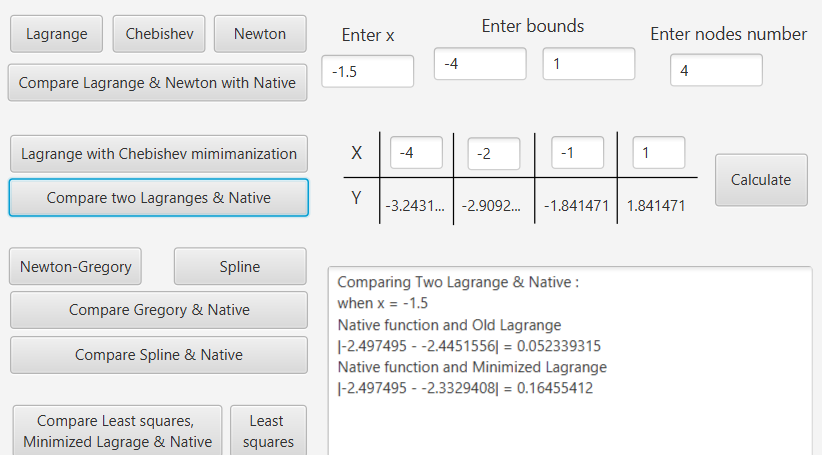


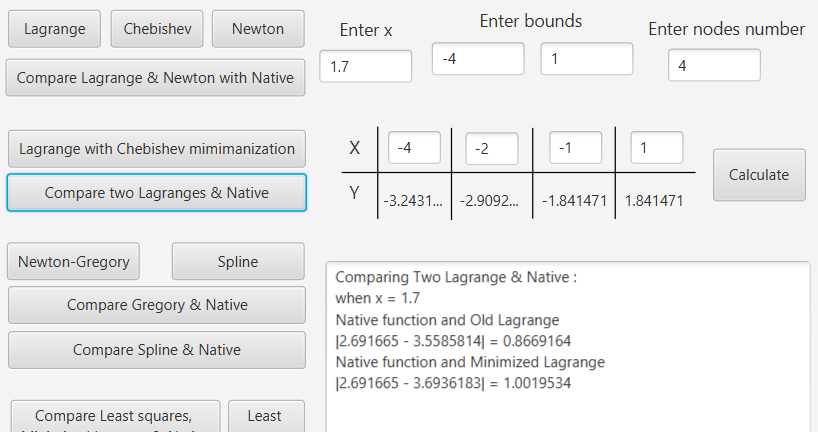


Завдання 2

Вузли Чебишева на заданому відрізку

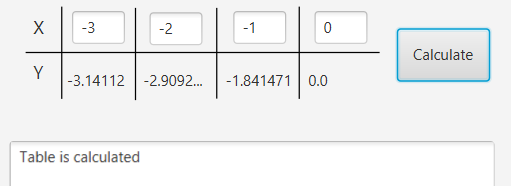
 Порівнювання значень поліномів Лагранжа і Лагранжа через вузли Чебишева з точним значенням



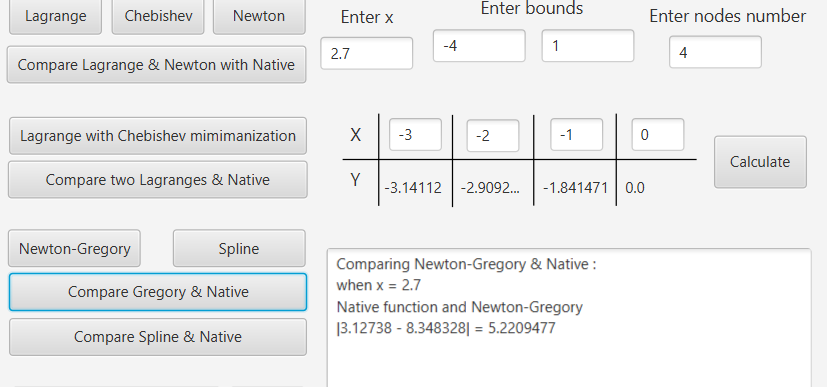


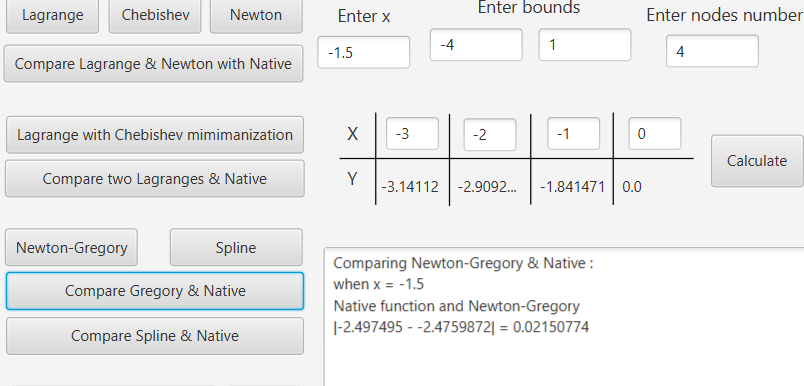
Завдання 3

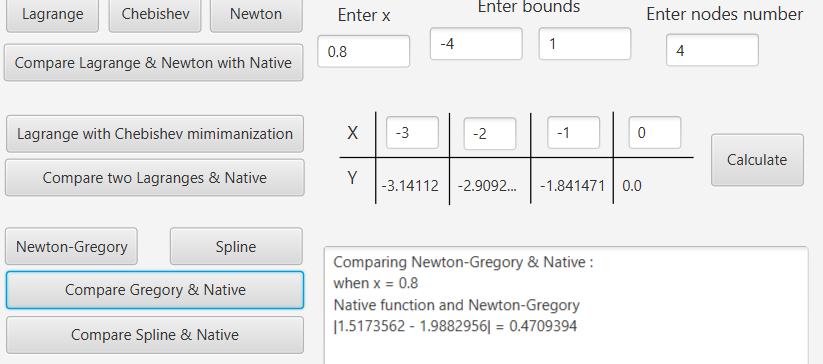
Нова таблиця



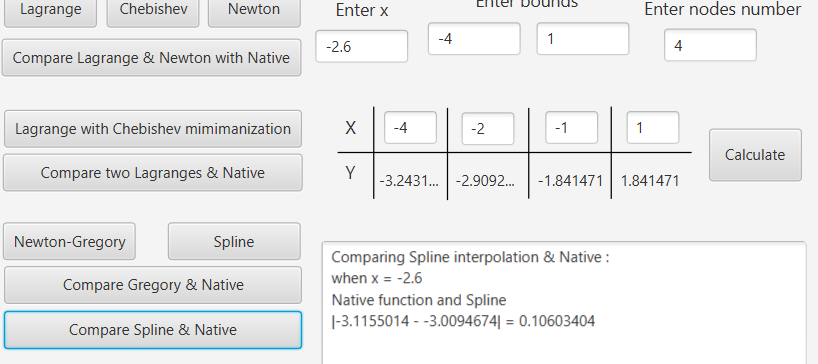
Порівнювання значень поліномів Ньютона-Грегорі з точним значенням

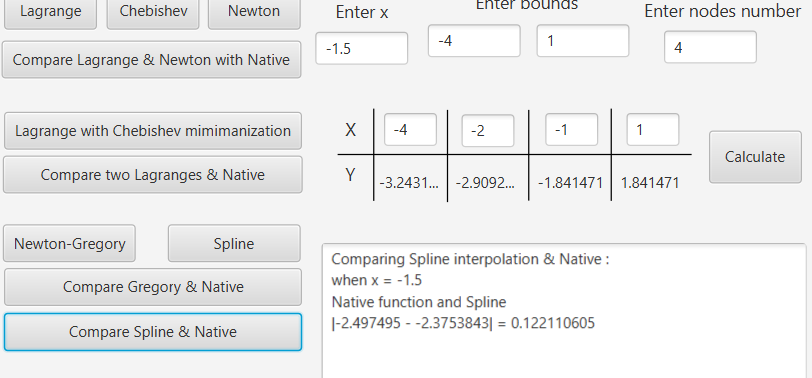
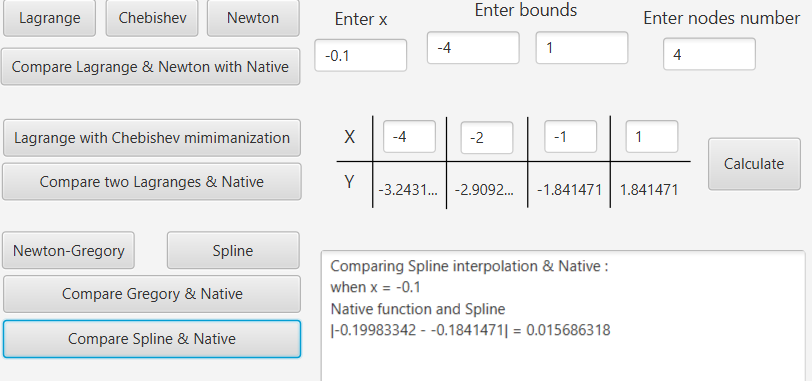




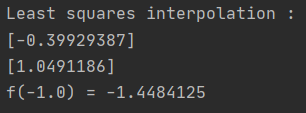


Завдання 4

Порівнювання значення лінійного сплайну з точним значенням



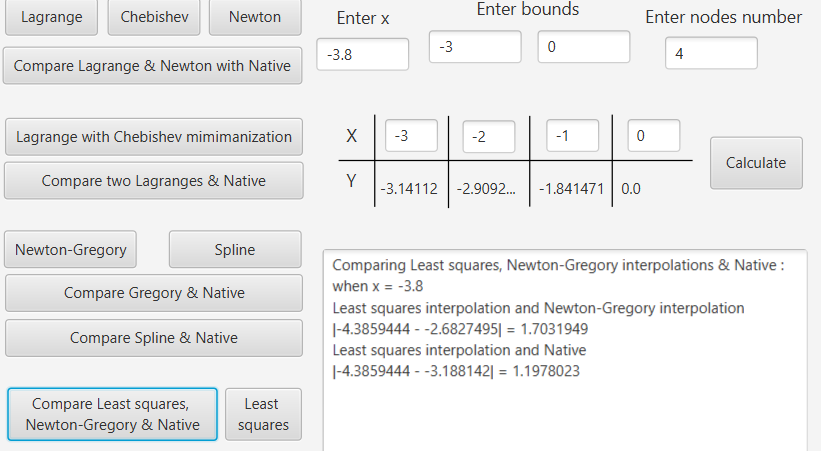
Завдання 5

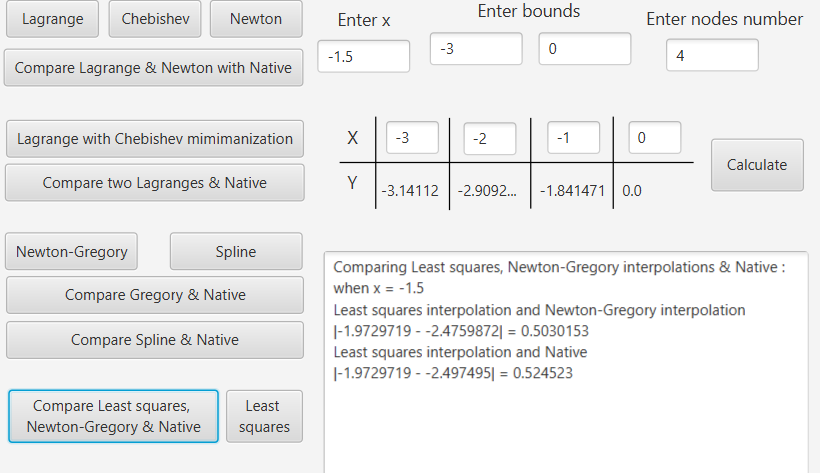
Коефіцієнти рівняння прямої :

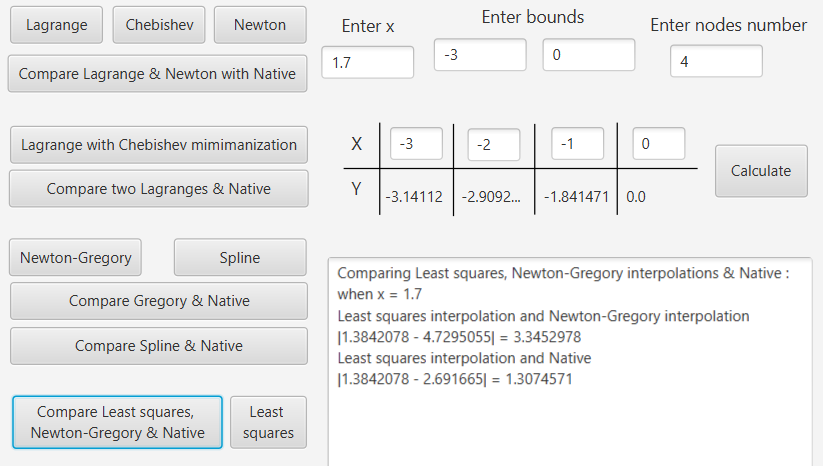
-0.3992 – це вільний член, а 1.049 -- це коефіцієнт при змінній

Графік Y = -0.3992 + 1.048x з точками таблиці 2.1

Порівнювання значень методу найменших квадратів з точним значенням і першим поліномом ньютона (для віддалених вузлів)

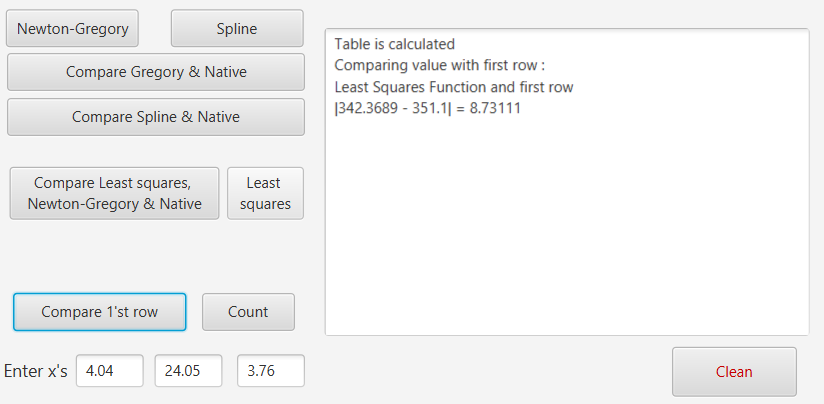






Завдання 6

Порівняння значення першого рядка зі значенням метода найменших квадратів в першому рядку



**Лістинг**

Main.java

package sample;  
  
import javafx.application.Application;  
import javafx.fxml.FXMLLoader;  
import javafx.scene.Parent;  
import javafx.scene.Scene;  
import javafx.stage.Stage;  
import java.util.Arrays;  
  
import static sample.Matrix.\*;  
  
  
  
public class Main extends Application {  
  
 @Override  
 public void start(Stage primaryStage) throws Exception{  
 Parent root = FXMLLoader.*load*(getClass().getResource("sample.fxml"));  
 primaryStage.setTitle("Hello World");  
 primaryStage.setScene(new Scene(root));  
 primaryStage.sizeToScene();  
 primaryStage.show();  
  
 }  
  
  
 public static void main(String[] args) {  
 *launch*(args);  
 }

}

Controller.java

package sample;  
  
  
import javafx.fxml.FXML;  
import javafx.scene.control.Button;  
import javafx.scene.control.Label;  
import javafx.scene.control.TextArea;  
import javafx.scene.control.TextField;  
  
import java.util.Arrays;  
  
import static sample.Methods.\*;  
import static sample.Reader.\*;  
  
public class Controller {  
  
 float[] ARR\_X = new float[4];  
 float[] ARR\_Y = new float[4];  
  
 @FXML  
 private Button BLagrange;  
 @FXML  
 private Button BNewton;  
 @FXML  
 private TextField Bx0;  
 @FXML  
 private Button BCompareLNN;  
 @FXML  
 private TextField FieldA;  
 @FXML  
 private TextField FieldB;  
 @FXML  
 private Button BChebishev;  
 @FXML  
 private Button BCreateNewLagrange;  
 @FXML  
 private Button BCompareTLN;  
 @FXML  
 private TextField FieldNodes;  
 @FXML  
 private TextField FieldX0;  
 @FXML  
 private TextField FieldX1;  
 @FXML  
 private TextField FieldX2;  
 @FXML  
 private TextField FieldX3;  
 @FXML  
 private Button BCalculate;  
 @FXML  
 private Label LabelY0;  
 @FXML  
 private Label LabelY1;  
 @FXML  
 private Label LabelY2;  
 @FXML  
 private Label LabelY3;  
 @FXML  
 private Button BNewtonGregory;  
 @FXML  
 private Button BCompareGN;  
 @FXML  
 private Button BSpline;  
 @FXML  
 private Button BCompareSN;  
 @FXML  
 private Button BSquares;  
 @FXML  
 private Button BCompareLGN;  
 @FXML  
 private Button BClean;  
 @FXML  
 private TextArea TextField = null;  
 @FXML  
 private Button BCount;  
 @FXML  
 private Button BCompareRow;  
 @FXML  
 private TextField x0;  
 @FXML  
 private TextField x1;  
 @FXML  
 private TextField x2;  
  
 @FXML  
 void initialize() {  
  
 CalculateTable();  
  
 BCalculate.setOnAction(event -> {  
  
 CalculateTable();  
  
 });  
  
 BLagrange.setOnAction(event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
  
 System.*out*.println("Lagrange interpolation : ");  
 TextField.setText(TextField.getText() + "Lagrange interpolation : \n");  
  
  
 float f = *LagrangeInterpolation*(ARR\_X, ARR\_Y, x);  
 System.*out*.println("f(" + x + ") = " + f);  
 TextField.setText(TextField.getText() + "f(" + x + ") = " + f + "\n");  
  
 });  
  
 BNewton.setOnAction(event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Newton interpolation : ");  
 TextField.setText(TextField.getText() + "Newton interpolation : " + "\n");  
  
  
 float f = *NewtonInterpolation*(ARR\_X, ARR\_Y, x);  
 System.*out*.println("f(" + x + ") = " + f);  
 TextField.setText(TextField.getText() + "f(" + x + ") = " + f + "\n");  
  
 });  
  
 BCompareLNN.setOnAction(Event -> {  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Comparing Lagrange, Newton & Native : ");  
 TextField.setText(TextField.getText() + "Comparing Lagrange, Newton & Native : \n");  
  
  
 float l = *LagrangeInterpolation*(ARR\_X, ARR\_Y, x);  
 float n = *NewtonInterpolation*(ARR\_X, ARR\_Y, x);  
 float f = *MyFunctionCalc*(x);  
  
 System.*out*.println("when x = " + x);  
 TextField.setText(TextField.getText() + "when x = " + x + "\n");  
  
 System.*out*.println("Native function and Lagrange\n|" + f + " - " + l + "| = " + Math.*abs*(f - l));  
 TextField.setText(TextField.getText() + "Native function and Lagrange\n|" + f + " - " + l + "| = "  
 + Math.*abs*(f - l) + "\n");  
  
 System.*out*.println("Native function and Newton\n|" + f + " - " + n + "| = " + Math.*abs*(f - n));  
 TextField.setText(TextField.getText() + "Native function and Newton\n|" + f + " - " + n + "| = "  
 + Math.*abs*(f - n) + "\n");  
  
 });  
  
 BChebishev.setOnAction(Event -> {  
  
 float[] arr = *ChebishevPoints*(Float.*parseFloat*(FieldA.getText()),  
 Float.*parseFloat*(FieldB.getText()), Integer.*parseInt*(FieldNodes.getText()) - 1);  
  
 System.*out*.println("Chebishev points : ");  
 TextField.setText(TextField.getText() + "Chebishev points : " + "\n");  
  
 System.*out*.println(Arrays.*toString*(arr));  
 TextField.setText(TextField.getText() + Arrays.*toString*(arr) + "\n");  
  
 });  
  
 BCreateNewLagrange.setOnAction(Event -> {  
  
  
 System.*out*.println("Minimized Lagrange interpolation : ");  
 TextField.setText(TextField.getText() + "Minimized Lagrange interpolation : " + "\n");  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 float f = *MinLagrangeInterpolation*(Float.*parseFloat*(FieldA.getText()),  
 Float.*parseFloat*(FieldB.getText()), Integer.*parseInt*(FieldNodes.getText()) - 1, x);  
  
 System.*out*.println("f(" + x + ") = " + f);  
 TextField.setText(TextField.getText() + "f(" + x + ") = " + f + "\n");  
  
  
 });  
  
 BCompareTLN.setOnAction(Event -> {  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Comparing Two Lagrange & Native : ");  
 TextField.setText(TextField.getText() + "Comparing Two Lagrange & Native : " + "\n");  
  
  
 float l = *LagrangeInterpolation*(ARR\_X, ARR\_Y, x);  
 float n = *MinLagrangeInterpolation*(Float.*parseFloat*(FieldA.getText()),  
 Float.*parseFloat*(FieldB.getText()), Integer.*parseInt*(FieldNodes.getText()) - 1, x);  
 float f = *MyFunctionCalc*(x);  
  
 System.*out*.println("when x = " + x);  
 TextField.setText(TextField.getText() + "when x = " + x + "\n");  
  
 System.*out*.println("Native function and Old Lagrange\n|" + f + " - " + l + "| = " + Math.*abs*(f - l));  
 TextField.setText(TextField.getText() + "Native function and Old Lagrange\n|" + f + " - " + l + "| = "  
 + Math.*abs*(f - l) + "\n");  
  
 System.*out*.println("Native function and Minimized Lagrange\n|" + f + " - " + n + "| = " + Math.*abs*(f - n));  
 TextField.setText(TextField.getText() + "Native function and Minimized Lagrange\n|" + f + " - " + n + "| = "  
 + Math.*abs*(f - n) + "\n");  
  
 });  
  
 BNewtonGregory.setOnAction(Event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Newton-Gregory interpolation : ");  
 TextField.setText(TextField.getText() + "Newton-Gregory interpolation : " + "\n");  
  
 float f = *NewtonGregoryInterpolation*(ARR\_X, ARR\_Y, x);  
 System.*out*.println("f(" + x + ") = " + f);  
 TextField.setText(TextField.getText() + "f(" + x + ") = " + f + "\n");  
  
 });  
  
 BCompareGN.setOnAction(Event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Comparing Newton-Gregory & Native : ");  
 TextField.setText(TextField.getText() + "Comparing Newton-Gregory & Native : " + "\n");  
  
  
 float g = *LagrangeInterpolation*(ARR\_X, ARR\_Y, x);  
 float f = *MyFunctionCalc*(x);  
  
 System.*out*.println("when x = " + x);  
 TextField.setText(TextField.getText() + "when x = " + x + "\n");  
  
 System.*out*.println("Native function and Newton-Gregory\n|" + f + " - " + g + "| = " + Math.*abs*(f - g));  
 TextField.setText(TextField.getText() + "Native function and Newton-Gregory\n|" + f + " - " + g + "| = "  
 + Math.*abs*(f - g) + "\n");  
  
 });  
  
 BSpline.setOnAction(Event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Spline interpolation : ");  
 TextField.setText(TextField.getText() + "Spline interpolation : " + "\n");  
  
 float f = *SplineInterpolation*(ARR\_X, ARR\_Y, x);  
 System.*out*.println("f(" + x + ") = " + f);  
 TextField.setText(TextField.getText() + "f(" + x + ") = " + f + "\n");  
  
 });  
  
 BCompareSN.setOnAction(Event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Comparing Spline interpolation & Native : ");  
 TextField.setText(TextField.getText() + "Comparing Spline interpolation & Native : " + "\n");  
  
 float s = *SplineInterpolation*(ARR\_X, ARR\_Y, x);  
 float f = *MyFunctionCalc*(x);  
  
 System.*out*.println("when x = " + x);  
 TextField.setText(TextField.getText() + "when x = " + x + "\n");  
  
 System.*out*.println("Native function and Spline\n|" + f + " - " + s + "| = " + Math.*abs*(f - s));  
 TextField.setText(TextField.getText() + "Native function and Spline\n|" + f + " - " + s + "| = "  
 + Math.*abs*(f - s) + "\n");  
  
 });  
  
 BSquares.setOnAction(Event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
 System.*out*.println("Least squares interpolation : ");  
 TextField.setText(TextField.getText() + "Least squares interpolation : " + "\n");  
  
  
 float s = *LeastSquares*(new float[][]{ARR\_X}, new float[][]{ARR\_Y}, new float[]{x});  
 System.*out*.println("f(" + x + ") = " + s);  
 TextField.setText(TextField.getText() + "f(" + x + ") = " + s + "\n");  
  
 });  
  
 BCompareLGN.setOnAction(Event -> {  
  
 float x = Float.*parseFloat*(Bx0.getText());  
  
  
 System.*out*.println("Comparing Least squares, Newton-Gregory interpolations & Native : ");  
 TextField.setText(TextField.getText() + "Comparing Least squares, Newton-Gregory interpolations & Native : "  
 + "\n");  
  
 float s = *LeastSquares*(new float[][]{ARR\_X}, new float[][]{ARR\_Y}, new float[]{x});  
 float n = *NewtonGregoryInterpolation*(ARR\_X, ARR\_Y, x);  
 float f = *MyFunctionCalc*(x);  
  
 System.*out*.println("when x = " + x);  
 TextField.setText(TextField.getText() + "when x = " + x + "\n");  
  
 System.*out*.println("Least squares interpolation and Newton-Gregory interpolation\n|" +  
 s + " - " + n + "| = " + Math.*abs*(s - n));  
 TextField.setText(TextField.getText() + "Least squares interpolation and Newton-Gregory interpolation\n|" +  
 s + " - " + n + "| = " + Math.*abs*(s - n) + "\n");  
  
 System.*out*.println("Least squares interpolation and Native\n|" + s + " - " + f + "| = " + Math.*abs*(s - f));  
 TextField.setText(TextField.getText() + "Least squares interpolation and Native\n|" +  
 s + " - " + f + "| = " + Math.*abs*(s - f) + "\n");  
  
 });  
  
 BClean.setOnAction(Event -> {  
 TextField.setText("");  
 });  
  
 BCount.setOnAction(Event -> {  
  
 float[] x = new float[]{Float.*parseFloat*(x0.getText()),  
 Float.*parseFloat*(x1.getText()), Float.*parseFloat*(x2.getText())};  
  
  
  
 System.*out*.println("Counting the Task : ");  
 TextField.setText(TextField.getText() + "Counting the Task : " + "\n");  
  
 float s = *LeastSquares*(*ReadX*(*WIDTH* , *LENGTH*, "E:\\Objects\\Modeling\\Lab1\\src\\sample\\data.txt"),  
 *ReadY*(*WIDTH* , *LENGTH*, "E:\\Objects\\Modeling\\Lab1\\src\\sample\\data.txt"), x);  
  
 System.*out*.println("f" + Arrays.*toString*(x) + " = " + s);  
 TextField.setText(TextField.getText() + "f" + Arrays.*toString*(x) + " = " + s + "\n");  
  
 });  
  
 BCompareRow.setOnAction(Event -> {  
  
 String path = "E:\\Objects\\Modeling\\Lab1\\src\\sample\\data.txt";  
  
 float[] x = *firstX*(*WIDTH*,path);  
 float y = *firstY*(*WIDTH*, path);  
  
 System.*out*.println("Comparing value with first row : ");  
 TextField.setText(TextField.getText() + "Comparing value with first row : " + "\n");  
  
 float s = *LeastSquares*(*ReadX*(*WIDTH* , *LENGTH*, path),  
 *ReadY*(*WIDTH* , *LENGTH*, path), x);  
  
 System.*out*.println("Least Squares Function and first row\n|" +  
 s + " - " + y + "| = " + Math.*abs*(s - y));  
  
 TextField.setText(TextField.getText() + "Least Squares Function and first row\n|" +  
 s + " - " + y + "| = " + Math.*abs*(s - y) + "\n");  
  
 });  
  
 }  
  
  
 void CalculateTable() {  
 ARR\_X[0] = Float.*parseFloat*(FieldX0.getText());  
 ARR\_X[1] = Float.*parseFloat*(FieldX1.getText());  
 ARR\_X[2] = Float.*parseFloat*(FieldX2.getText());  
 ARR\_X[3] = Float.*parseFloat*(FieldX3.getText());  
  
 LabelY0.setText(String.*valueOf*(ARR\_Y[0] = *MyFunctionCalc*(ARR\_X[0])));  
 LabelY1.setText(String.*valueOf*(ARR\_Y[1] = *MyFunctionCalc*(ARR\_X[1])));  
 LabelY2.setText(String.*valueOf*(ARR\_Y[2] = *MyFunctionCalc*(ARR\_X[2])));  
 LabelY3.setText(String.*valueOf*(ARR\_Y[3] = *MyFunctionCalc*(ARR\_X[3])));  
  
 System.*out*.println("Table is calculated");  
  
 TextField.setText(TextField.getText() + "Table is calculated\n");  
 }  
  
}

Methods.java

package sample;  
  
import java.util.Arrays;  
  
import static sample.Matrix.\*;  
  
public class Methods {  
  
 static float LagrangeInterpolation(float[] arrX, float[] arrY, float x) {  
 float result = 0;  
  
 for (int i = 0; i < arrX.length; i++) {  
 float P = 1;  
  
 for (int j = 0; j < i; j++) P \*= (x - arrX[j]) / (arrX[i] - arrX[j]);  
 for (int j = i + 1; j < arrX.length; j++) P \*= (x - arrX[j]) / (arrX[i] - arrX[j]);  
  
 result += P \* arrY[i];  
 }  
  
 return result;  
 }  
  
 static float MyFunctionCalc(float x) {  
 return (float) (Math.*sin*(x) + x);  
 }  
  
 static float NewtonInterpolation(float[] arrX, float[] arrY, float x) {  
  
 float sum = arrY[0];  
 for (int i = 1; i < arrX.length; i++) {  
  
 float F = 0;  
 for (int j = 0; j <= i; j++) {  
  
 float div = 1;  
 for (int k = 0; k <= i; k++)  
 if (k != j) div \*= (arrX[j] - arrX[k]);  
  
 F += arrY[j] / div;  
 }  
  
 for (int k = 0; k < i; k++)  
 F \*= (x - arrX[k]);  
 sum += F;  
 }  
 return sum;  
  
 }  
  
 static float[] ChebishevPoints(float a, float b, int n) {  
  
 float[] point = new float[n + 1];  
  
 for (float i = 0; i <= n; i++)  
 point[(int) i] = (float) ((a + b) / 2 + ((b - a) / 2) \* Math.*cos*(((2 \* i + 1) / (2 \* n + 2)) \* Math.*PI*));  
  
  
 return point;  
  
 }  
  
 static float MinLagrangeInterpolation(float a, float b, int n, float x) {  
  
 float[] arrX;  
 arrX = *ChebishevPoints*(a, b, n);  
  
 float[] arrY = new float[arrX.length];  
 for (int i = 0; i < arrX.length; i++) arrY[i] = *MyFunctionCalc*(arrX[i]);  
  
 return *LagrangeInterpolation*(arrX, arrY, x);  
 }  
  
 static float NewtonGregoryInterpolation(float[] arrX, float[] arrY, float x) {  
  
 float h = Math.*abs*(arrX[0] - arrX[1]);  
  
 for (int i = 0; i < arrX.length - 2; i++)  
 if (h != Math.*abs*(arrX[i + 1] - arrX[i + 2])) {  
 try {  
 throw new Exception("Can't be used! H isn't a constant");  
 } catch (Exception e) {  
 e.printStackTrace();  
 return 0;  
 }  
 }  
  
 float sum = 0;  
 float q = (x - arrX[0]) / h;  
  
 float[][] diff = new float[arrX.length][arrX.length];  
 for (int i = 0; i < arrX.length; i++) diff[0][i] = arrY[i];  
  
  
 for (int i = 1; i < arrX.length; i++)  
 for (int j = 0; j < arrX.length - i; j++)  
 diff[i][j] = diff[i - 1][j + 1] - diff[i - 1][j];  
  
  
 for (int i = 0; i < arrY.length; i++) {  
  
 float F = diff[i][0];  
  
 for (int j = 0; j < i; j++) {  
 F \*= (q - j) / (j + 1);  
 }  
  
 sum += F;  
 }  
  
 return sum;  
 }  
  
 static float SplineInterpolation(float[] arrX, float[] arrY, float x) {  
  
 for (int i = 0; i < arrX.length; i++) {  
 if (x >= arrX[i] && x <= arrX[i + 1]) {  
  
 float a = (arrY[i + 1] - arrY[i]) / (arrX[i + 1] - arrX[i]);  
 float b = arrY[i] - a \* arrX[i];  
  
 return a \* x + b;  
 }  
 }  
 System.*err*.println("Spline is OUT OF BORDER!");  
 return 0;  
 }  
  
 static float LeastSquares(float[][] arrX, float[][] arrY, float[] x) {  
  
  
 float[][] matX = new float[arrX.length + 1][arrX[0].length];  
 for (int i = 0; i < arrX.length + 1; i++)  
 for (int j = 0; j < arrX[0].length; j++) {  
 matX[i][j] = (i == 0) ? 1 : arrX[i - 1][j];  
 }  
  
  
 float[][] matY = *TransponateMatrix*(arrY);  
  
 float[][] b = *MultiplyMatrix*(matX, *TransponateMatrix*(matX));  
  
 b = *ReversedMatrix*(b);  
 b = *MultiplyMatrix*(b, matX);  
 b = *MultiplyMatrix*(b, matY);  
  
 for (int i = 0; i < b.length; i++) System.*out*.println(Arrays.*toString*(b[i]));  
  
 float sum = b[0][0];  
 for (int i = 0; i < x.length; i++) sum += x[i] \* b[i + 1][0];  
  
  
 return sum;  
 }  
  
}

Matrix.java

package sample;  
  
import java.util.Arrays;  
  
public class Matrix {  
  
 static float[][] TransponateMatrix(float[][] arr) {  
  
 float[][] arrT = new float[arr[0].length][arr.length];  
 for (int i = 0; i < arr.length; i++)  
 for (int j = 0; j < arr[0].length; j++)  
 arrT[j][i] = arr[i][j];  
  
 return arrT;  
 }  
  
 static float[][] MultiplyMatrix(float[][] arr1, float[][] arr2) {  
  
 float[][] arr = new float[arr1.length][arr2[0].length];  
  
 for (int i = 0; i < arr1.length; i++)  
 for (int j = 0; j < arr2[0].length; j++) {  
 arr[i][j] = 0;  
 for (int k = 0; k < arr2.length; k++)  
 arr[i][j] += arr1[i][k] \* arr2[k][j];  
  
 }  
  
  
 return arr;  
 }  
  
 static float DeterminateMatrix(float[][] arr) {  
  
 if (arr.length != arr[0].length) try {  
 throw new Exception("Matrix isn't square!");  
 } catch (Exception e) {  
 e.printStackTrace();  
 return 0;  
 }  
  
 if (arr.length == 1) return arr[0][0];  
 int l = arr.length - 1;  
 float num = 0;  
  
 for (int i = 0, j = 0; i < arr.length; i++) {  
  
  
 float[][] temp = new float[l][l];  
  
 for (int k = 0, n = 0; n < l \* l; k++) {  
 if (!(k / arr.length == i || k % arr.length == j))  
 temp[n / l][n++ % l] = arr[k / arr.length][k % arr.length];  
 }  
  
 float d = arr[i][j] \* *DeterminateMatrix*(temp);  
 if ((i + j) % 2 == 1) d \*= -1;  
 num += d;  
  
 }  
  
 return num;  
 }  
  
 static float[][] ReversedMatrix(float[][] arr) {  
  
 float det = 1 / *DeterminateMatrix*(arr);  
  
 float[][] arrT = *TransponateMatrix*(arr);  
  
 float[][] arrR = *AllyMatrix*(arrT);  
  
 return *MultiplyMatrix*(arrR, det);  
 }  
  
 static float[][] MultiplyMatrix(float[][] arr, float det) {  
  
 for (int i = 0; i < arr.length; i++)  
 for (int j = 0; j < arr.length; j++)  
 arr[i][j] \*= det;  
  
 return arr;  
  
 }  
  
 static float[][] AllyMatrix(float[][] arr) {  
  
 float[][] arrA = new float[arr.length][arr.length];  
  
 for (int i = 0; i < arr.length; i++)  
 for (int j = 0; j < arr.length; j++) {  
  
 int l = arr.length - 1; // 2  
 float[][] temp = new float[l][l];  
  
 for (int k = 0, n = 0; n < l \* l; k++) {  
 if (!(k / arr.length == i || k % arr.length == j))  
 temp[n / l][n++ % l] = arr[k / arr.length][k % arr.length];  
 }  
  
 arrA[i][j] = *DeterminateMatrix*(temp);  
 if ((i + j) % 2 == 1) arrA[i][j] \*= -1;  
  
 }  
 return arrA;  
 }  
  
  
}

Reader.java

package sample;  
  
import java.io.File;  
import java.io.FileNotFoundException;  
import java.util.Arrays;  
import java.util.Scanner;  
  
public class Reader {  
  
 static int *LENGTH* = 41;  
 static int *WIDTH* = 4;  
  
  
 static float[][] ReadX(int width, int len, String path) {  
 File file =  
 new File(path);  
 Scanner sc = null;  
 try {  
 sc = new Scanner(file);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 float[][] arrX = new float[width - 1][len];  
  
 for (int i = 0; sc.hasNextLine(); i++) {  
 if (!(i % width == (width - 1)))  
 arrX[i % width][i / width] = Float.*parseFloat*(sc.next());  
 else sc.next();  
  
 }  
  
 return arrX;  
 }  
  
 static float[][] ReadY(int width, int len, String path) {  
 File file =  
 new File(path);  
 Scanner sc = null;  
 try {  
 sc = new Scanner(file);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 float[][] arrY = new float[1][len];  
  
 for (int i = 0; sc.hasNextLine(); i++) {  
 if (!(i % width == (width - 1))) sc.next();  
 else arrY[0][i / width] = Float.*parseFloat*(sc.next());  
  
 }  
  
 return arrY;  
 }  
  
 static float firstY(int width, String path) {  
 File file =  
 new File(path);  
 Scanner sc = null;  
 try {  
 sc = new Scanner(file);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
  
 for (int i = 0; i < width-1; i++){  
 sc.next();  
 }  
  
 return Float.*parseFloat*(sc.next());  
 }  
  
 static float[] firstX(int width, String path) {  
 File file =  
 new File(path);  
 Scanner sc = null;  
 try {  
 sc = new Scanner(file);  
 } catch (FileNotFoundException e) {  
 e.printStackTrace();  
 }  
 float[] f = new float[width-1];  
  
 for (int i = 0; i < width-1; i++){  
 f[i] = Float.*parseFloat*(sc.next());  
 }  
  
 return f;  
 }

}

**Висновки**

Під час цієї лабораторної роботи були вивчені різні методи апроксимації, такі як : поліномом Лагранжа(з та без вузлів Чебишева), поліномом Ньютона(для рівновіддалених вузлів та нерівних проміжків), методом лінійних сплайнів та методом найменших квадратів. Були порівняні способи на практиці в конкретних точках. Також був побудований апроксимований графік функції за допомогою методу найменших квадратів